Bottom-up Reuse Guidelines: Packaging

Bottom-up Reuse Guidelines: Software Packaging Technical Note

by Ananth Rao (SGT / NASA GSFC)

Introduction

Planning for effective software packaging is an essential part of the software development cycle and in general should not be left as an afterthought, when deployment is being considered. An important consideration when developing software with re-use in mind (irrespective of software type i.e prototype or production quality) is, that at some point, different flavors of the underlying operating system will need to be used for deployment, and this cannot be avoided. This implies that differences in run-time environments such as libraries, header files, file system layout etc. need to be considered.

This document outlines some general factors to consider when packaging re-usable software components for deployment in heterogeneous environments. Packaging in this context is mostly relevant to Unix based systems, where, it is assumed that the vast majority of back-end processing takes place, though many of the concepts are also applicable to desktop PC-based systems.

The need for effective deployment planning

As already mentioned, deployment planning is better served if it is done early in the development cycle. This allows for application development, with a view to making these applications portable across operating systems. This entails using standardized interfaces with external packages that may already have been ported to a wide variety of platforms, as well as using standardized portable interfaces with the underlying operating system such as POSIX.

Goals of an effective packaging system

Many automatic tools are already available for packaging software systems for deployment. While these tools vary widely, based on the underlying language (e.g Perl, Java, C etc) and type of component (e.g. re-usable library or module, J2EE Enterprise Bean, or independent application) being deployed, some of the overall goals to be considered before selecting the appropriate tool, include:

- *Ease of use* -- Ease of use of the tool is of primary importance. Users of the tool consist of both package creators (developers) as well as package deployers (administrators or end-users).
- *Out-of-the-box configurability* -- During deployment, the package should be usable with a minimum of user interaction. The tool should be capable of being configured to automate many of the tasks specific to the packaging lifecycle, such as auto-detecting operating system versions as well as locations of standard libraries and header files.
- *Dependency management capability* -- Many applications are dependent on one or more other applications or libraries, some of which may be version specific. The tool should be capable of being configured to detect the existence (or non-existence) of required dependencies.
- *Upgrade consistency* -- Once deployed, this software will need to be maintained through the application of patches and upgrades. Consideration should be give to see if the same package management system can be used for this purpose.

Packaging Lifecycle

Most deployable software systems follow similar lifecycles. In general, when building software packages for deployment, it is useful to itemize the tasks involved, based on the following classification:

- Pre-installation tasks
- Dependency evaluation
- Software installation [install | update | remove]
- Post-installation tasks
- Verifying Integrity

This classification ensures that deployments are reproducible across platforms as well as forcing package creators to consider alternate deployment configurations such as deployment in unprivileged environments.

Suggested Tools

C/C++ Applications

Packages for deployment come in two flavors:

- source code packages, containing the source code and the configuration files necessary to produce a binary package, and

- binary packages that contain pre-built binaries for specified operating system environments

The Red Hat Package Manager (RPM) is a system that can be used to build, install, query, verify, update and remove individual software packages. A software package in this context, consists of an archive of files and associated metadata, used to install (or update, or remove) these files. The metadata includes helper scripts and other descriptive information about the package. RPM can be used with both source-code as well as binary packages when used with source-code packages, it is suggested that it be used in conjunction with GNU Config, as described below.

The use of GNU Config is suggested for applications that need to be deployed, when source code is packaged.

The paradigm of:

```
./configure  make  make test# optional  make install
```

is used with the vast majority of C language based applications that need to be compiled and linked on a diverse set of heterogeneous Unix platforms. The Configure script first queries the underlying operating system and build sub-system (compiler, linker etc) for system related configuration items that need to be known and builds the Makefile based on this information. The system is then compiled and installed without any further user input. Details on how to package systems using the configure script are available at [GNU Config].

OpenPKG unifies the use of RPM with GNU Config, in addition to providing other value added features for large scale heterogeneous deployment.

Perl Modules

Very similar to the GNU config system, is Perl's Makemaker system. Here a "Makefile.PL" perl script is used to auto-generate a Makefile. The 'Makefile.PL' script first sets default locations for the installation, and then refers to the MANIFEST file. The MANIFEST is a list of all files that are a part of the project. When Makefile.PL is run, it examines MANIFEST, and checks that all the files required for the project are present. The install process then consists of the following steps:

```
perl Makefile.PL  make  [make test]# optional  make install
```

Java Applications

Apache Ant is the de-facto standard for open-source Java projects. It is an extensible, platform independent build tool that can be used to automate all phases of a Java projects life-cycle, from compilation and testing to installation and deployment. Ant is used in conjunction with an XML based configuration file that contains a set of different targets for the build process. In this respect, the functionality of Ant is much like that of Makefiles in that it can be used to track source code dependencies and automatically compile them if needed, but goes far beyond this and is specifically tailored to Java environments.

References

- RPM http://rpm.org. The premier open-source packaging environment for Linux environments. Provides sophisticated querying capabilities.
- GNU Config - http://www.gnu.org/software/autoconf. Used in a large number of open-source packages to auto-detect OS environments differences and automatically build appropriate configuration files for source code builds.
- OpenPKG http://www.openpkg.org. Built on top of RPM and provides value added features to the base system.
- OSSP - http://www.ossp.org. Pre-built Unix re-usable components for application development. These components are used by OpenPKG.
- Apache Ant - http://ant.apache.org/. The de-facto standard for building and packaging Java projects.